AMENDMENTS TO THE CLAIMS

1. (Currently amended) A method for performing write-before-read interlocks for recovery unit operands, comprising: [[~~holding up operands of R-unit registers for a minimum number of cycles until all prior updates have completed by comparing addresses of said R-unit registers in at least one queue and interlocking valid matches of said R-unit register addresses , the method comprising:~~]]

receiving a plurality of recovery unit[[~~R-unit~~]] register addresses associated with a millicode architecture environment, said recovery unit[[~~R-unit~~]] register addresses including at least one of a millicode general register and a millicode access register;

storing said recovery unit[[~~R-unit~~]] register addresses in a plurality of queues including a write queue, a pre-write queue, and a read queue;

accessing said queues;

comparing said recovery unit[[~~R-unit~~]] register addresses;

determining matches between said recovery unit[[~~R-unit~~]] register addresses; and

implementing one or more write-before-read interlocks after said determining produces a valid match, said one or more write-before-read interlocks being implemented until said comparing is no longer active[[~~,~~]];

wherein[[~~whereby~~]] operands of recovery unit[[~~R-unit~~]] registers are updated during an operand prefetching period[[~~ with a minimum number of cycles~~]].

2. (Previously presented) The method of claim 1 wherein one of said write-before-read interlocks causes a millicode read instruction not to execute.

3. (Canceled)

4. (Canceled)

5. (Currently amended) The method of claim 1[[~~3~~]] wherein said comparing includes comparing said recovery unit[[~~R-unit~~]] register addresses sent to said read queue against said recovery unit[[~~R-unit~~]] register addresses sent to said write queue.

6. (Currently amended) The method in claim 1[[3]] wherein said determining includes matching valid recovery unit[[~~R-unit~~]] register addresses of said write queue and said read queue.

7. (Currently amended) The method in claim 6[[3]] wherein said determining also includes matching said valid recovery unit[[~~R-unit~~]] register addresses of said pre-write queue and said read queue.

8. (Canceled)

9. (Previously presented) The method in claim 1 wherein said one or more write-before-read interlocks prevent millicode read instructions from being processed.

10. (Currently amended) The method in claim 1 wherein a write queue accumulates said recovery unit[[~~R-unit~~]] register addresses.

11. (Previously presented) The method in claim 1 wherein said updating occurs when an SRAM receives the accumulated results from a write queue.

12. (Currently amended) A system for performing write-before-read interlocks for recovery unit operands, comprising:[[~~holding up operands of R-unit registers for a minimum number of cycles until all prior updates have completed by comparing addresses of said R-unit registers in at least one queue and interlocking valid matches of said R-unit register addresses, the system comprising:~~]]

a plurality of queues for storing recovery unit[[~~R-unit~~]] register addresses associated with a millicode architecture environment, said recovery unit[[~~R-unit~~]] register addresses including at least one of a millicode general register and a millicode access register, the queues including a write queue, a pre-write queue, and a read queue;

a comparator for comparing said recovery unit[[~~R-unit~~]] register addresses in said plurality of queues and determining matches between said recovery unit[[~~R-unit~~]] register addresses; and

a plurality of write-before-read interlocks that are implemented after valid matches of said recovery unit[[~~R-unit~~]] register addresses are determined, said write-before-read interlocks being implemented until said comparing is no longer active~~,~~;

wherein[[~~whereby~~]] operands of recovery unit[[~~R-unit~~]] registers are updated during an operand prefetching period[[~~with a minim number of cycles~~]].

13. (Previously presented) The system of claim 12 wherein one of said write-before-read interlocks causes a millicode read instruction not to execute.

14. (Canceled)

15. (Canceled)

16. (Currently amended) The system of claim 12[[~~14~~]] wherein said comparator compares said recovery unit[[~~R-unit~~]] register addresses sent to said read queue against said recovery unit[[~~R-unit~~]] register addresses sent to said write queue.

17. (Currently amended) The system in claim 15 wherein said comparator determines said valid recovery unit[[~~R-unit~~]] register address matches between said write queue and said read queue.

18. (Currently amended) The system in claim 15 wherein said comparator determines said valid recovery unit[[~~R-unit~~]] register address matches between said pre-write queue and said read queue.

19. (Canceled)

20. (Previously presented) The system in claim 13 wherein said interlocks prevent millicode read instructions from being processed.

21. (Currently amended) The system in claim 14 wherein said recovery unit[[~~R-unit~~]] addresses are accumulated in said write queue.

22. (Currently amended) The system in claim 14 wherein said recovery unit[[~~R-unit~~]] registers are updated when an SRAM receives the accumulated results from said write queue.

23. (New) The method of claim 11, wherein the SRAM has at least one port, the priority of the port being changed from reading to writing, when one of the write-before-read interlocks is active.

24. (New) The method of claim 22, wherein the SRAM has at least one port, the priority of the port being changed from reading to writing, when one of the write-before-read interlocks is active.